Straight or Skew?
A new puzzle

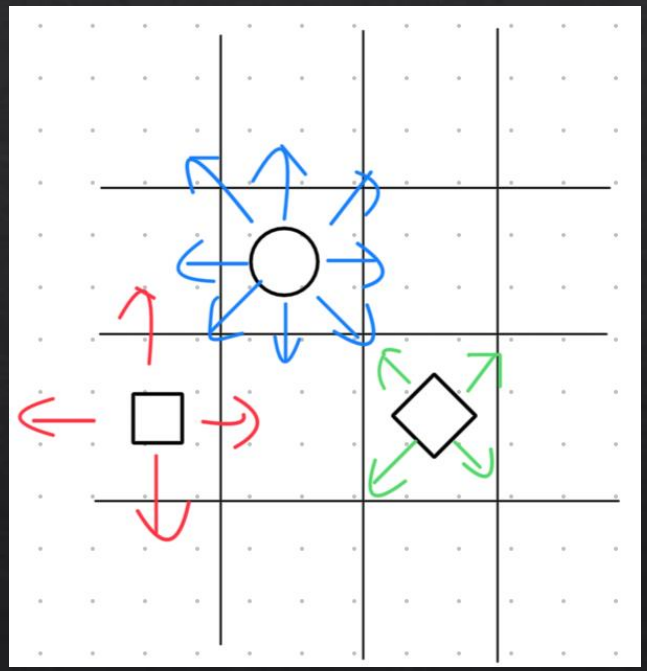
Zhi (Richie) Li , NetID: zhil7

Outline

- Introduction
- Demo and Play
- Methods
- Algorithm Analysis
- Performance Measurement
- ~~Game Balance~~
- Summary and Future Works

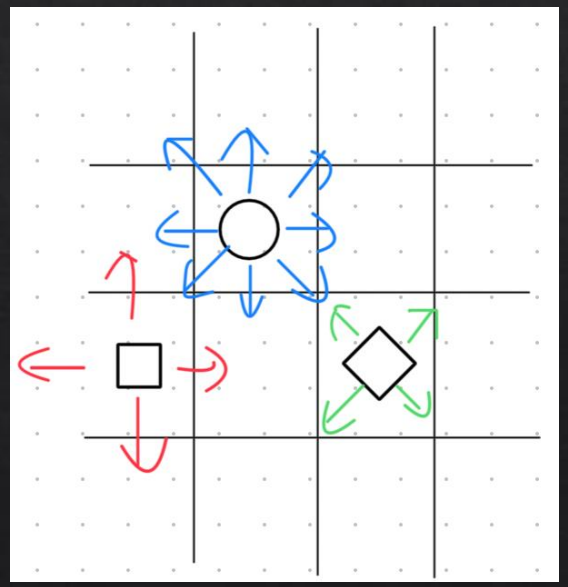
Introduction

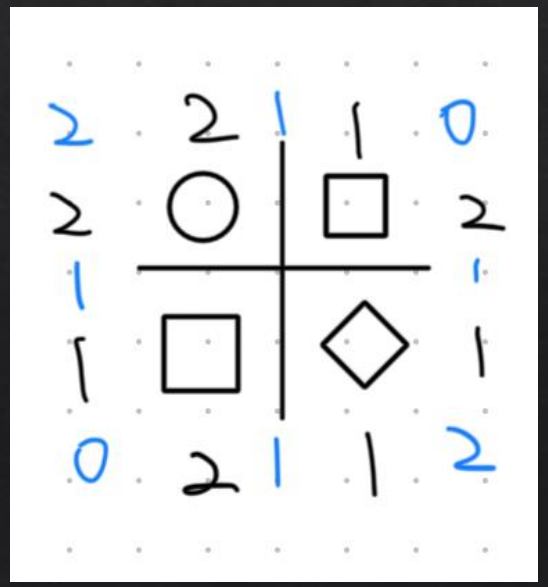
- This is a game that I might have invented myself. I was inspired by the Play Station controller, which contains different shapes of buttons, and also by lots of square grid puzzles.
- Players play on an NXN grid with numbers on the sides, just as we did in "Undead", but now players place squares, diamonds, and circles in the cells. A square can be seen straight, a diamond can be seen diagonally, and a circle can be seen in either direction, as shown in the following picture.

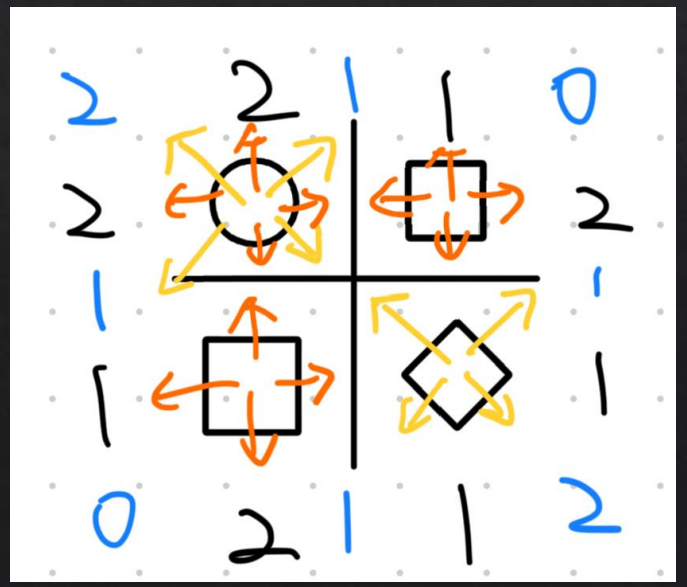


Introduction

- Note that instead of **4n** views (n=length of the grid), we have **8n** views because we now have the skewed direction that ends differently than the straight ones. As shown in the third picture, the orange arrows point to black numbers, and the yellow arrows point to blue numbers.





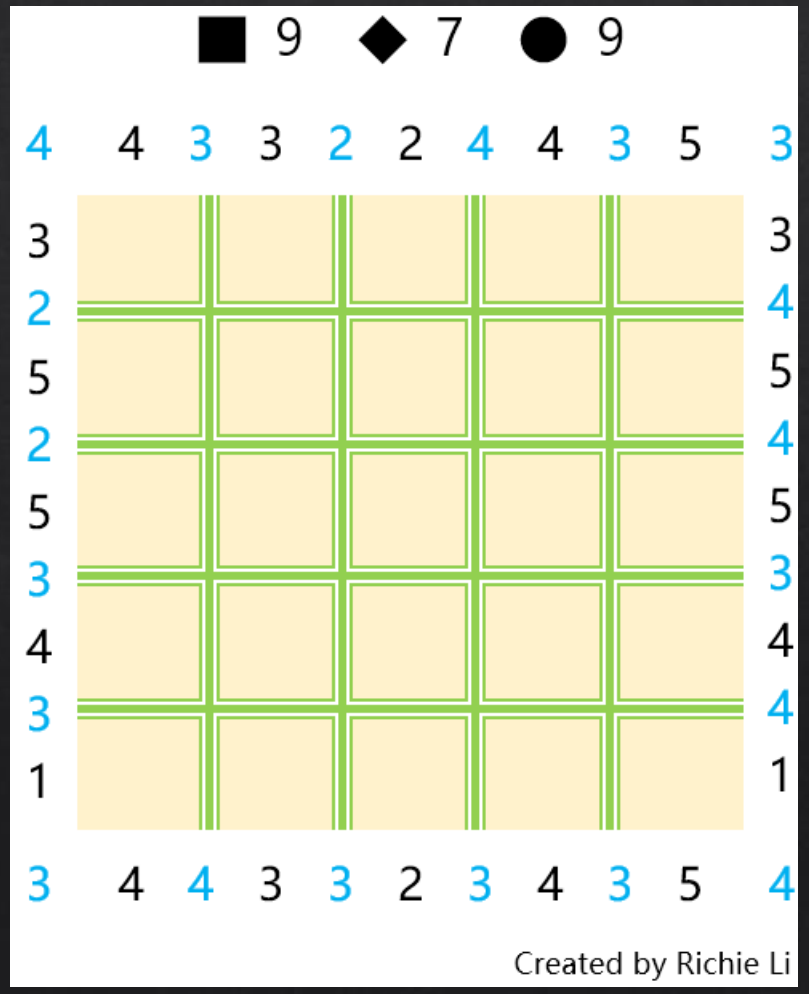


Demo and Play

- Let me show the game on my iPad
- You will also be playing one later!

Methods – Data Structures

- Counter: dictionary
- Clues: lists in each direction
  - Straight (2)
  - Oblique (4)
- Grid: 2D object list
  - Cell: row, col, shape



Methods – Algorithms

- What do we need to calculate when solving or generating the puzzle?
- General
  - Is the board full?
  - Place a shape into the grid
  - Is the puzzle valid?
  - Modifying the clues

Methods – Algorithms

- Solving:
  - Finding Solutions
    - What cell should we choose to place which shape, brute force*
  - Backtracking
    - If we did not successfully place a shape, go back and retry the last move, recursively
- Generating:
  - Randomly pick shapes to form a unvalidated puzzle
    - Solve it to see if we should regenerate

Algorithm Analysis

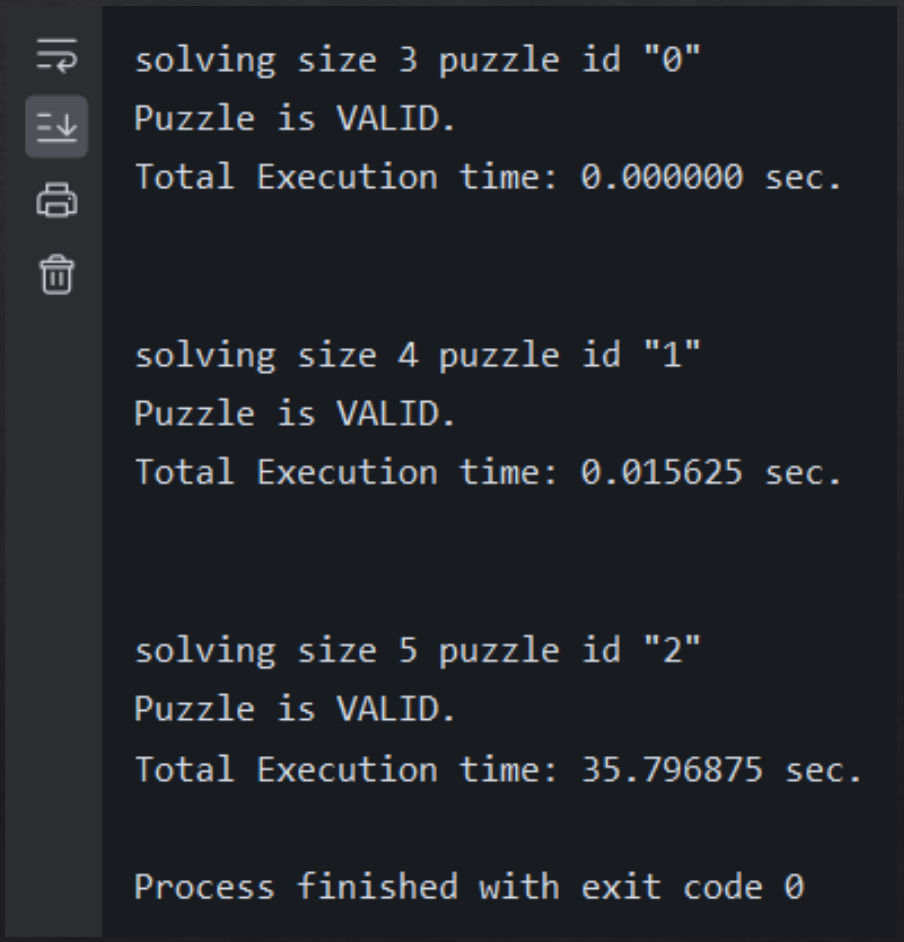
- **def is_valid(self, Cell) -> bool:**
  - Checks if a placement is valid
  - Modifies the shape of a cell (as if we were putting a shape into the grid)
  - Add 1 to edge clues that are affected
    - Square, Diamond: 4 edges
    - Circle: 8 edges
  - Time complexity: O(4n) or O(8n) ~ O(n); n = size of the grid

Algorithm Analysis

- **def find_solutions(self) -> int:**
  - Finds the number of solution(s) of the puzzle, breaks when over 1
  - Places a valid move to the grid, if no more valid shapes to place, backtrack
  - Time complexity: O(k^{n^2}); k = kinds of shapes, n = size of the grid
    - For example, a 4x4 and 5x5 will differ by 3^9=19683 times (0.006s to 120s!)
    - (Very sad performance, needs to add some predeterministics before brute force)

Performance Measurement

- Comparing the solving speed of 3x3, 4x4, 5x5
- 3x3 and 4x4 are solved immediately, while
- 5x5 took half a minute!!
- Whose taking up the time?



Performance Measurement

Statistics | Call Graph

Name	Call Count		Time (ms)		Own Time (ms)	
is_full	12090680	12m	24538	19.1%	24538	19.1%
backtrack	6045344	6m	55402	43.0%	21338	16.6%
is_valid_puzzle	12292245	12m	36497	28.3%	20905	16.2%
find_cell_obliques	10505841	10m	16247	12.6%	14415	11.2%
get_next_empty_cell	6045341	6m	12152	9.4%	12152	9.4%
find_solutions	3		128796	100.0%	12080	9.4%
modify_oblique_clue	7497478	7m	10535	8.2%	10535	8.2%
place_shape	12292245	12m	45767	35.5%	9270	7.2%
<built-in method builtins.abs>	21011682		1832	1.4%	1832	1.4%
<built-in method builtins.len>	6045370		581	0.5%	581	0.5%
<method 'pop' of 'list' objects>	6045341		580	0.5%	580	0.5%
<method 'append' of 'list' objects>	6045353		564	0.4%	564	0.4%
Generator	1		0	0.0%	0	0.0%
Board	1		0	0.0%	0	0.0%
__init__	3		0	0.0%	0	0.0%

Performance Measurement

Statistics | Call Graph

Name	Call Count	Time (ms)	Own Time (ms)
backtrack	6045344	56025 46.0%	21561 17.7%
is_valid_puzzle	12292245	37138 30.5%	21196 17.4%
find_cell_obliques	10505841	16560 13.6%	14657 12.0%
is_full	12090680	15442 12.7%	13436 11.0%
get_next_empty_cell	6045341	12931 10.6%	12931 10.6%
find_solutions	3	121882 100.0%	12477 10.2%
modify_oblique_clue	7497478	10749 8.8%	10749 8.8%
place_shape	12292245	46372 38.0%	9233 7.6%
<built-in method builtins.len>	30226739	2558 2.1%	2558 2.1%
<built-in method builtins.abs>	21011682	1903 1.6%	1903 1.6%
<method 'pop' of 'list' objects>	6045341	594 0.5%	594 0.5%
<method 'append' of 'list' objects>	6045353	581 0.5%	581 0.5%
<lambda>	1	0 0.0%	0 0.0%
find_spec	1	0 0.0%	0 0.0%
__init__	2	0 0.0%	0 0.0%

After optimizing the is_full function..
24.538s → 13.436s (11.102s faster) in a total runtime of 120s

Summary and Future Works

- Coming up with an original puzzle wasn't that hard, but it was difficult to design a balanced one
  - The need of classifying which is easy and which is hard (Future Work)
- The diagonal lines might not be intuitive as it seems
- As we did in undead, the generation time of a bigger puzzle rises a lot when increasing size
  - So I stored the created puzzle for later use
  - Come up with a better solver other than brute force solving
- Maybe creating a GUI to make it easier to let users access the game
  - For example, they can click to insert shapes or click "GIVE ME A CLUE, IT'S TOO HARD"

References

- Undead from Simon Tatham's Portable Puzzle Collection
  - www.chiark.greenend.org.uk/~sgtatham/puzzles/js/undead.html
- Previous Assignments
- Documents of Python modules

Thank you for your attention